A gentle intro to Golang and the Go-universe

Alex-P. Natsios

GoCode.Thessaly(7)

28 feb 2014

The Language

A few words about GO

Go, also called **golang**, was initially developed at google in 2007 and announced in 2009 but most of its popularity boost (and hype) came much later with its first stable release (1.0) on 28 Mar 2012.

Characteristics:

- Compiled
- Statically typed and garbage collected
- Object Oriented (but not in the usual way)
- Sane Concurrency
- Fast Compilers
- Rich Standard Library
- Scalable Tools

The Gopher!

The Classic Example

```
package main

import "fmt"

func main() {
        fmt.Println("Hello, World!")
}
```

Packages

- Go code lives in packages.
- A package consists of one or more source(.go) files.
- Typically all files belonging to a package are placed in the same directory.
- Visibility is determined by case: Foo is exported, foo is not.

Declarations

```
import "fmt"
const answer = 42
var something [10]byte
var names = []string {"Alex", "George", "Maria" }
```

Types

- All basic types you would expect to have.
- But string is a basic type.
- No pointer arithmetic.
  New (or not exactly that new) types:
- Slices: Much like many other modern languages, []int
- Maps: Because suddenly everybody needs them, map[string]int
- Interfaces: for Polymorphism, interface{}
- Channels: Used to communicate with goroutines, chan int

Scope

- Universe
- Package
- File (for imports)
- Function
- Block

Concurrency

Concurrency is a property of systems in which several computations are executing simultaneously and potentialy interacting with each other.
WARNING: Concurrency is NOT parallelism, although it enables parallelism if you have a multiprocessor system.

Goroutines

- Independently executing functions, launched by a **go** statement.
- *NOT* threads.
- VERY cheap, you might have thousands of goroutines running under the same thread.
- Have their own dynamic call stack (growning and shrinking as needed).

Channels

A channel provides a connection between goroutines, allowing communication.
Channels can be unbuffered or buffered, so they both communicate and synchronize.
Buffered channels are asynchronous.

```
package main

import "fmt"

func main() {
greetings := make(chan string, 2)

go func() {
greetings <- "Hello"
greetings <- "World!"
}()

greet1 := <-greetings
greet2 := <-greetings
fmt.Println(greet1, greet2)
}
```

When in main(), "<-greetings" is executed it waits for a value to be sent.
Same goes for our anonymous function that expects a reciever to be in place in order for the greetings to be sent.
If no sender/receiver is ready (they both must be) then we wait until they are!

The Toolchain

The go tool

go build - To compile the package.
go get - To resolve and install deps.
go test - To run the test suite and benchmarks.
go install - To install the package.
go doc - To generate documentation.
go fmt - To properly format your code.
go run - To build and run the app.
go tool - To run extra tools.
and more (properly "integrated" in the **go** tool)

Building and workspaces

A **GO** program can be compiled and linked without additional build info. A single tool can compile either individual files or entire systems.
In order to work without build scripts a certain directory structure **MUST** be followed.

workspace

```
workspace/
  bin/
  pkg/
  src/
```

workspaces
cont.

Creating a workspace:

```
mkdir -p $HOME/GoCode/{bin, pkg, src}
```

Telling go about it:

```
export GOPATH="$HOME/GoCode"
export PATH="$PATH:$GOPATH/bin"
```

Links

Golang homepage:

`golang.org`

Go tour:

`tour.golang.org`

Package Doc:

`golang.org/pkg`

A little outdated but still useful page:

`go-lang.cat-v.org`

Obligatory subreddit:

`reddit.com/r/golang/`

Thank you for your attention!

Alex-P. Natsios

drakevr@2f30.org

http://drakevr.gr

http://www.linkedin.com/in/drakevr

http://www.github.com/drakevr

http://www.facebook.com/drakevr

http://www.twitter.com/drakevr